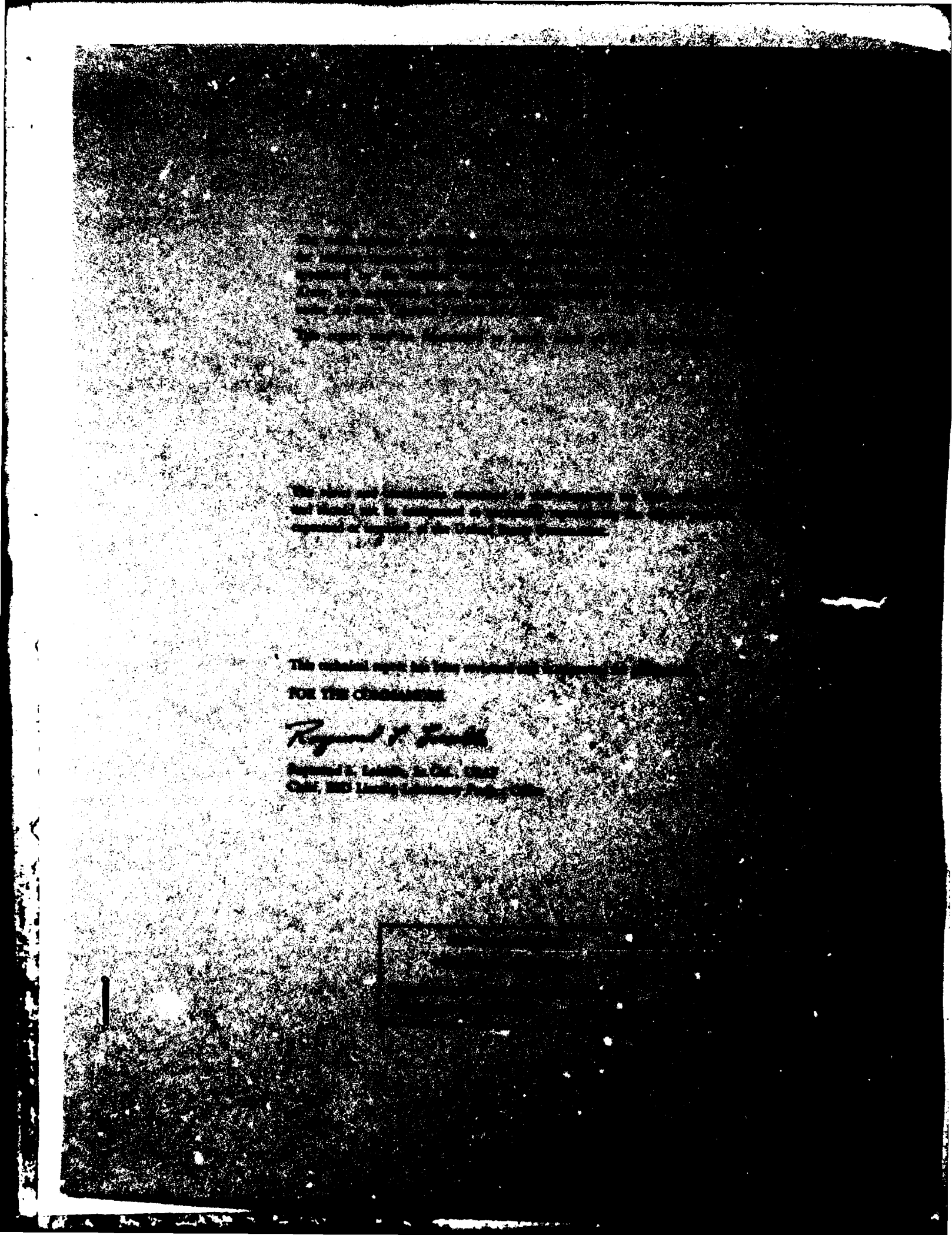
AD-A104 858 MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB F/G 12/1
PRACTICAL ASPECTS OF NONLINEAR OPTIMIZATION,(U)
JUN 81 R B HOLMES, J W TOLLESON F19628-80-C-0002
UNCLASSIFIED TR-576 ESD-TR-81-155 NL

END
DATE FILMED
08 81
DTIC



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

PRACTICAL ASPECTS OF NONLINEAR OPTIMIZATION

R.B. HOLMES
J.W. TOLLESON
Group 32

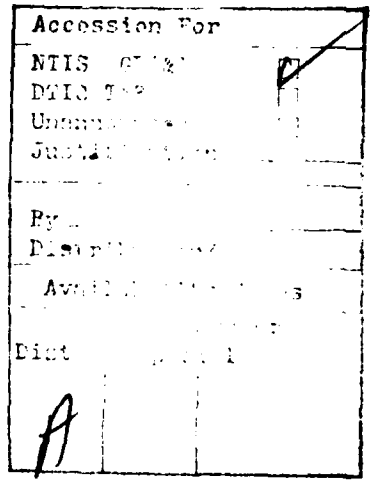
TECHNICAL REPORT 576

19 JUNE 1981

LEXINGTON MASSACHUSETTS

ABSTRACT

A general purpose nonlinear programming method and computer code are presented. The method is basically heuristic but extremely simple and reliable. Interactive operation of the code and its performance on several test problems is described.



CONTENTS

I. BACKGROUND

Quite commonly, problems in the statistical and engineering sciences require the optimization of some function for their solution. Such problems include those of resource allocation, portfolio selection, curve and surface fitting (e.g., linear or nonlinear regression), approximation theory, signal processing algorithms (optimizing a model fit), and optimal control (via discretization). In addition, systems of nonlinear equations can be solved by minimizing some function of the magnitudes of the residuals. As for the statistical sciences in general we have the following recent comment of C.R. Rao: "All statistical procedures are, in the ultimate analysis, solutions to suitably formulated optimization problems" [1].

Throughout this report we shall be concerned with optimizations of the particular form:

$$\text{minimize } f(x), \quad x \in \Omega, \tag{1}$$

where Ω is a solid subset of some finite dimensional Euclidean space, describable by finitely many contraints. More precisely, we assume that Ω can be expressed as

$$\Omega = \left\{ x \in R^n : \; a_i \le x_i \le b_i, \; 1 \le i \le n, \; c_j \le g_j(x) \le d_j, \; 1 \le j \le m \right\}, \tag{2}$$

with the understanding the Ω so defined has a non-empty interior (is "solid"). No qualitative assumptions on the objective

function f in (1) or on the constraint functions g_j in (2), such as convexity, differentiability, etc., are made.

Problems of the form of (1) are known as mathematical programs. They have been the subject of an immense amount of study over the last forty years, beginning with the well-known special case of linear programs, wherein the constraint and objective functions are linear functions of their variables. As a result a considerable body of both theory and computational algorithms has been developed, usually under convexity and/or differentiability assumptions. The books [2-8] provide a representative description of these developments; in addition, there are specialized journals such as Journal of Optimization Theory and Applications (since 1967) and Mathematical Programming (since 1970).

There is a basic dichotomy in programming algorithms: they may be designed to converge to local or global minima. Recall that a local minimum of f is a point $x_0 \in \Omega$ such that $f(x) \geq f(x_0)$ for every x in Ω that is sufficiently near to x_0, while a global minimum is indeed a solution of (1). Local minima can be characterized or at least partially identified in various ways, depending on the nature of the problem. A prototypical necessary condition for x_0 to be a local minimum of a differentiable program (subject to a "constraint qualification" on the geometry of Ω) is due to Kuhn and Tucker [9] and might be considered to be the

fundamental theorem of mathematical programming. In its geometric form it states in essence that the negative gradient of f at x_0 must belong to the cone generated by the gradients at x_0 of the active constraints there. Further, under various generalized convexity hypotheses, e.g., that f should be pseudoconvex and the g_j quasiconvex, any local minimum is actually a global minimum; in such a case, then, the Kuhn-Tucker condition characterizes the solutions of (1). This remark applies a fortiori to the case where all functions are actually convex.

Such characterizations are sometimes used to provide stopping criteria for numerical algorithms. For example, should a local minimum occur at an interior point of Ω, as would certainly be the case if there were no constraints, the K-T condition reduces to the vanishing of the gradient ∇f of f there, and in practice an algorithm might be terminated at a point where $||\nabla f||$ becomes sufficiently small. Of course, such a point will not necessarily be a local minimum nor even close to one in the absence of convexity conditions. Similarly the popular "method of feasible directions" [3,10] attempts to terminate at a Kuhn-Tucker point.

In the important special case where all constraint functions are linear, so that Ω is a simplex, the constraint qualification is automatically satisfied. In this case several effective algorithms exist provided that ∇f is available. These include the method of feasible directions, the gradient projection algorithm [11], and an approximation method wherein f is replaced

by its first order Taylor expansion; this approximation yields a linear program and the algorithm generates a sequence of such programs [12].

In this linearly constrained case there are very efficient finite algorithms available provided the objective function f is either linear or else convex and quadratic. Failing this, but with ∇f available, the various methods mentioned in the preceding paragraph generate sequences of linear programs, systems of linear equations, and line searches (one-dimensional optimizations) to be solved. Yet other methods generate sequences of quadratic programs to be solved by constructing quadratic approximations to f [13, 14]. When ∇f is not available but can be assumed to exist and be smooth, a powerful but rather involved algorithm has recently been described in [15].

For the general (nonlinearly) constrained mathematical program there are adaptations of the above algorithms obtained by linearizing the contraints. There is also a variety of indirect methods which may operate outside the feasible set Ω. Examples are cutting plane algorithms and the use of penalty functions [2, 4]. The former generate a sequence of linear programs over a decreasing sequence of simplices which enclose Ω; the possiblity of doing so depends on a standard trick of utilizing a linear objective function at the cost of introducing an additional variable and constraint. The latter are functions that are large off Ω and which are added to f; a sequence of these new objective functions is then minimized without constraints. All these

methods involve assorted numerical difficulties as well as the possiblity of termination outside Ω. Should the objective function be undefined or meaningless off Ω, this outcome is naturally untenable.

In the last decade there has been upsurge of interest in the global optimization problem per se; see [16, 17, 18]. This problem in full generality is very different from and more difficult than the problem of local optimization just discussed. There is relatively little theory in support of proposed methods unless fairly strong restrictions are placed on the problems to which they are applied.

The methods may roughly be classified as deterministic or probabilistic, although many methods have both features. Most of the effort has gone into unconstrained optimization or else into problems which only involve bounds on the variables.

Of these the probabilistic methods appear in general more attractive, in part because of the lack of hypotheses that must be imposed on the problem structure and in part because they seem less sensitive to increases in dimension. Another difference concerns the manner of validation of particular algorithms: with probabilistic methods it may be possible to give a theoretical a priori measure of the probability of "success", while for a deterministic algorithm such a probability can at best be estimated only through extensive testing.

Among the several popular probabilistic methods we mention specifically only two: the multistart method and the sample function method. The former is essentially an iterative cycle of random searches throughout Ω followed by application of a local minimization algorithm. The latter hypothesizes that the objective function f is a sample function of a known stochastic process on Ω. The statistics of the process then permit new points in Ω to be generated according to the information provided by the values of f at preceding points. Further statistical testing can be employed to assess the likelihood that f could indeed be a sample path of such a process. Usually the Wiener process is assumed. At this date the theory and computational requirements seem to limit this method to small (perhaps only 1!) dimensional problems.

To conclude this brief background we note that a general summary of computer codes for mathematical programming that have been tested, documented, and are available to the public occurs in [19]. Somewhat earlier there appeared a collection of FORTRAN listings of optimization codes, along with brief descriptions of the algorithms and their operations [20]. The potential user is left to make his own choice as to which method will best serve his purpose and, unfortunately, the various codes are not of uniformly high quality.

II. METHODOLOGICAL AND PROGRAMMING DESIDERATA

The authors' experience with numerous "real-world" optimization problems has persuaded them of the need for a simple reliable optimization routine that could be interactively operated. Such a program should in particular be applicable to an essentially arbitrary objective function, about which little or no prior information may be available, save a procedure for its evaluation. This situation occurs, for example, when the objective function represents the output of some "black box" operation, perhaps from a process optimization or simulation model. In any event the nature of such a function cannot be well understood, gradients are unavailable except via finite difference approximations, and convexity (occasionally even continuity) is in doubt. It is also often the case that the feasible region must be strictly respected, in that nonfeasible points have no significance for the problem.

We feel that the nonavailability of analytic gradients precludes the use of the various local search methods mentioned in Section I. While some codes have been developed that utilize difference approximations, this is in general well known to be a numerically risky and often unstable procedure. There are the twin perils of too large a differencing interval (leading to truncation errors) and too small an interval (leading to round-off and cancellation errors). The situation is not hopeless

[15, 21, 22] but it is subtle and one we think worth avoiding, if possible.

The requirements of simplicity, robustness, and generality lead us to a combination of global random search and local pattern search. This last term describes a class of methods wherein the objective function is systematically evaluated at several points in a neighborhood of a current point, the resulting function values are compared, and a new improved point is selected. A finite number of failures to locate an improved point results in termination. These methods, and their termination criteria, are for the most part heuristic, unsupported by significant mathematical statements of optimality. Nevertheless, they can usually be expected to perform well, if suboptimally; they will rather quickly produce a "good" function value, although extreme accuracy may require many further evaluations, and premature stalls are possible. Exact solutions to specialized problems (such as an extreme point solution of a concave minimization) will not normally be determined precisely.

These last observations lead to an important caveat: if a program with special structure is confronted there will most likely exist a more effective specialized algorithm. Of course tracking down such a method, especially in reliable coded form, may not be easy (the sources in [19, 20] should be helpful for this purpose). Another practical point is this: computer time, even

at several hundred dollars/hour is still cheaper than programmer/analyst time at $10-20/hour. Even if, say, a routine utilizing gradient (and possible Hessian) data is available, these derivatives must still be calculated and encoded. The savings might typically amount to a few tenths of a second of execution time, probably not a cost effective trade-off. An important possible exception to this advice occurs when the purpose of the optimizing code is to serve as a subroutine of some large program, which is to process data in real time, and to apply some control law. In such a case speed and efficiency are likely to be of real concern, and specialized software may be required.

A review of several popular methods of pattern search, including the complex method which underlies the approach described below, is given in Chapter VII of [4] and in [6,7]. We also take note of an earlier two-stage heuristic method for unconstrained optimization announced in [23]. It consists of a single non-random global search of pre-determined length from a given initial point followed by a local pattern search. The latter is of the Hooke-Jeeves type [6,7], with a modification to account for curvature in the gradient path. No restart or other interactive features are offered.

III. THE PROPOSED METHOD

The following method meets the criteria we have just imposed: it is simple, versatile, robust, and can be operated interactively. It requires as inputs the absolute minimum of information that is needed to specify a mathematical program of the type (1), (2), plus a few parameters that essentially determine how much work the program will do. We also emphasize that the proposed method requries no linear algebraic subroutines, such as matrix inversions or linear equation solvers, nor does it construct derivative approximations. Our code can thus serve as an "off the shelf" package for optimization, model fitting, and nonlinear equation solving. Several examples illustrating the program's use for such applications will be given in Section V.

In essence the program utilizes the fundamental idea of Box [24], wherein a random finite set of feasible points is generated and successively deformed (based on information gleaned from function evaluations) until it collapses around a local minimum, or else (on occasion) becomes mired in an area where the function is essentially constant. Several modifications of this basic iteration have been made to prevent various possible traps and infinite loops that are possible when the feasible set fails to be convex or even connected. Further modifications have been made to provide several starting and continuation options for the users, as well as printing and termination options.

To describe the program in greater detail it is convenient to treat separately its three major aspects:

1) the basic iteration;

2) termination;

3) initialization.

We will then make some comments about the continuation and printing options

Referring to the notation of (1), and assuming that $\Omega \subset R^n$, we are given a finite subset K of Ω, termed a "complex". Typically the size of K (denoted KPTS in the program) is strictly between n and 2n; unless otherwise instructed our program sets KPTS = least integer $\geq$ 1.5n. Three points of K are now singled out: XLO, XNXTHI, XHI, so that

$$f(XLO) = \min\{f(X): X \in K\}$$

$$\leq \ldots \leq f(XNXTHI) \leq$$

$$f(XHI) = \max\{f(X): X \in K\}.$$

The centroid XC of $\{X \in K: X \neq XHI\}$ is now computed and, for the moment, is assumed to be feasible (i.e., XC $\in \Omega$). Next the point XHI is reflected through XC to a point XNU so that dist (XNU, XC) = STEP·dist(XC, XHI); here STEP is a program parameter typically in the range (1,2]. Thus

$$XNU = XHI + (1+STEP)\ (XC-XHI). \qquad (3)$$

Now if it happens that XNU is both feasible and satisfies f(XNU) < f(XNXTHI) we define a new complex by replacing in K the point XHI by XNU; in this case the basic iteration is complete.

Should either of the two requirements on XNU fail, XNU is backed halfway towards the centroid XC along the line through XC and XHI. If this point meets both requirements we substitute it for XHI and obtain a new complex. If not, we halve again. If after a finite number (typically 5-10) of such attempts we are still unsucessful, we set XNU = XC and commence moving XNU halfway towards XLO, again trying to meet the twin requirements of feasibility and lower f-value. If a finite number (typically 10-20) of such attempts are still unsuccessful, the algorithm is terminated if the last move resulted in feasibility only. If not, we reflect the latest XNU an equal distance through XLO and if both requirements still cannot be met, the algorithm is again terminated. In all these cases of premature termination a warning message is printed indicating either an infeasible direction (and suggesting that the best point XLO is located in a very "thin" region of Ω), or else the inability to decrease the function value sufficiently, in which case the algorithm is simply "stuck".

There is one other conceivable difficulty that can arise during a basic iteration, and that occurs when the centroid XC is not feasible. Of course, this could only happen when Ω is not convex. In this case we again set XNU = XC and commence

moving XNU towards XLO as just described, repeating exactly the above steps. Hence, either a new complex is obtained or else termination ensues with the appropriate warning message.

Earlier variants of this basic iteration are described in the original source [24] and in [4, 20]. A flow chart of our version appears in Fig. 1 (wherein the identification Point←→XNU should be made).

In most "reasonable" optimization problems the basic iteration as just described will result in a new complex. Consequently we can expect to generate a sequence of complexes and so must decide when to cease. Roughly we should do so when no further progress is being achieved. In practice we evaluate this condition by means of the absolute and relative oscillation of f on the complexes. If either of these quantities drops under given thresholds for a finite number (typically 5) of complexes in succession, the algorithm terminates. Otherwise a new basic iteration is carried out. Of course an upper bound on the number of iterations must also be supplied.

Thus, assuming that new complexes can be generated we control termination with four program parameters: ABSTOL, RELTOL, NTOL, and NLOOPS. If for a sequence of length NTOL of complexes

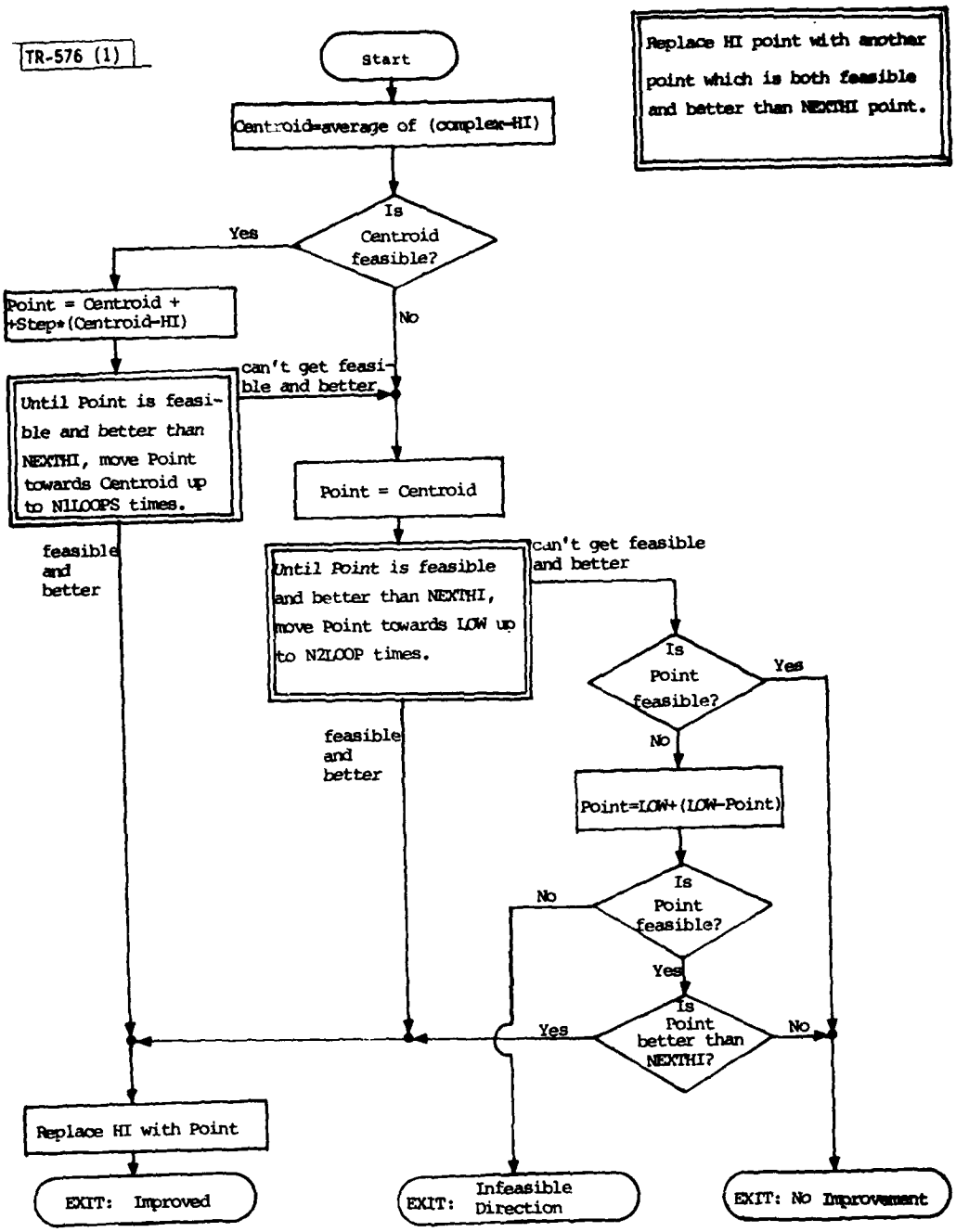


Fig. 1. The basic iteration.

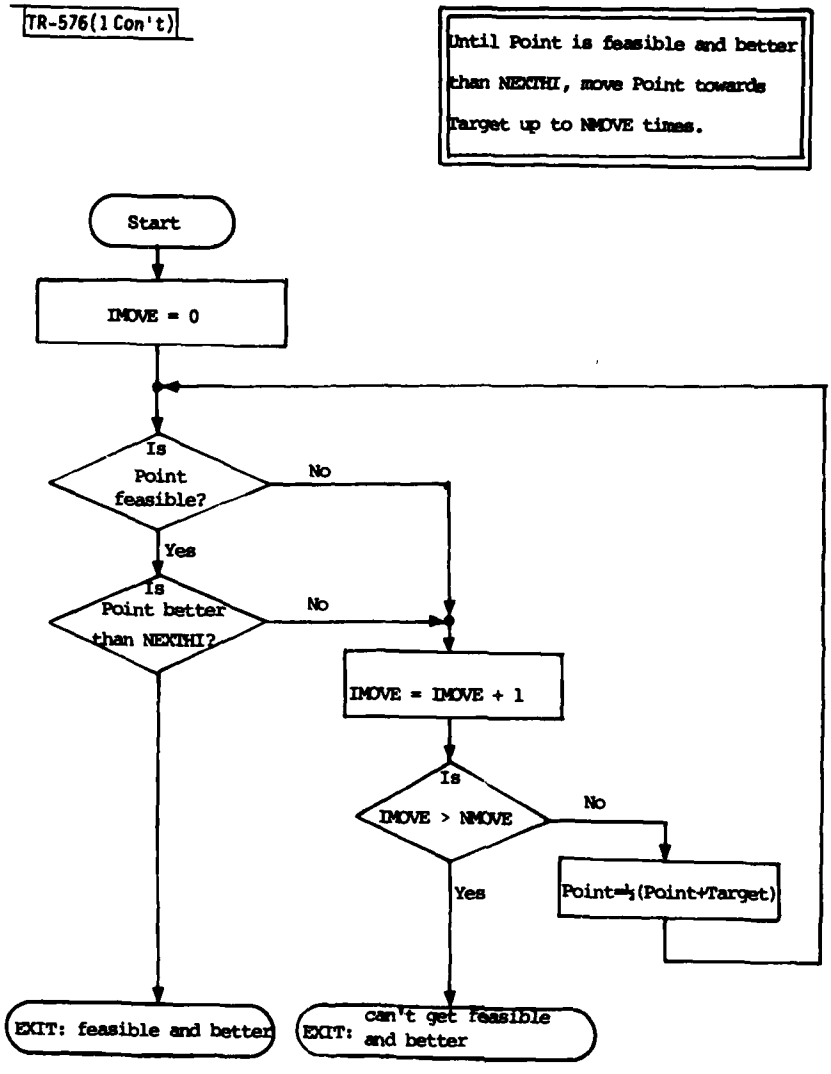


Fig. 1. Continued.

K_{i+1}, K_{i+2},..., K_{i+NTOL} we find

$$f(XHI) - f(XLO) \leq ABSTOL, \tag{4}$$

or failing this, if

$$f(XHI) - f(XLO) \leq RELTOL \cdot |f(XHI)|, \tag{5}$$

the program terminates with a convergence assertion. Otherwise, it terminates with a warning when the number of basic iterations exceeds NLOOPS. These termination procedures are displayed in Fig. 2.

To initialize the optimization process we must construct a first complex. Our program permits this to be done in three ways:

a) by direct assignment;

b) randomly about a given feasible point;

c) choosing the best KPTS points from a preliminary random search of Ω.

Option a) might be used when the geometric structure of Ω was known and simple (e.g., $\Omega=\{(x_1,\ldots,x_n): 0\leq x_i, \sum x_i \leq 1\}$, and no information about the minimum of f over Ω was available. Option b) is valuable when there is a priori information about f to the extent that we believe that for some $x^{(0)} \in \Omega$, $f(x^{(0)})$ is close to min $f(\Omega)$. The remaining KPTS-1 points for the initial complex are then produced sequentially by first drawing a point randomly within the region defined by the explicit constraints $\{x: a_i \leq x_i \leq b_i,\ 1 \leq i \leq n\}$ in (2), and then retracting it by halves towards $x^{(0)}$ until it becomes feasible. If a finite number (typically 10-20)

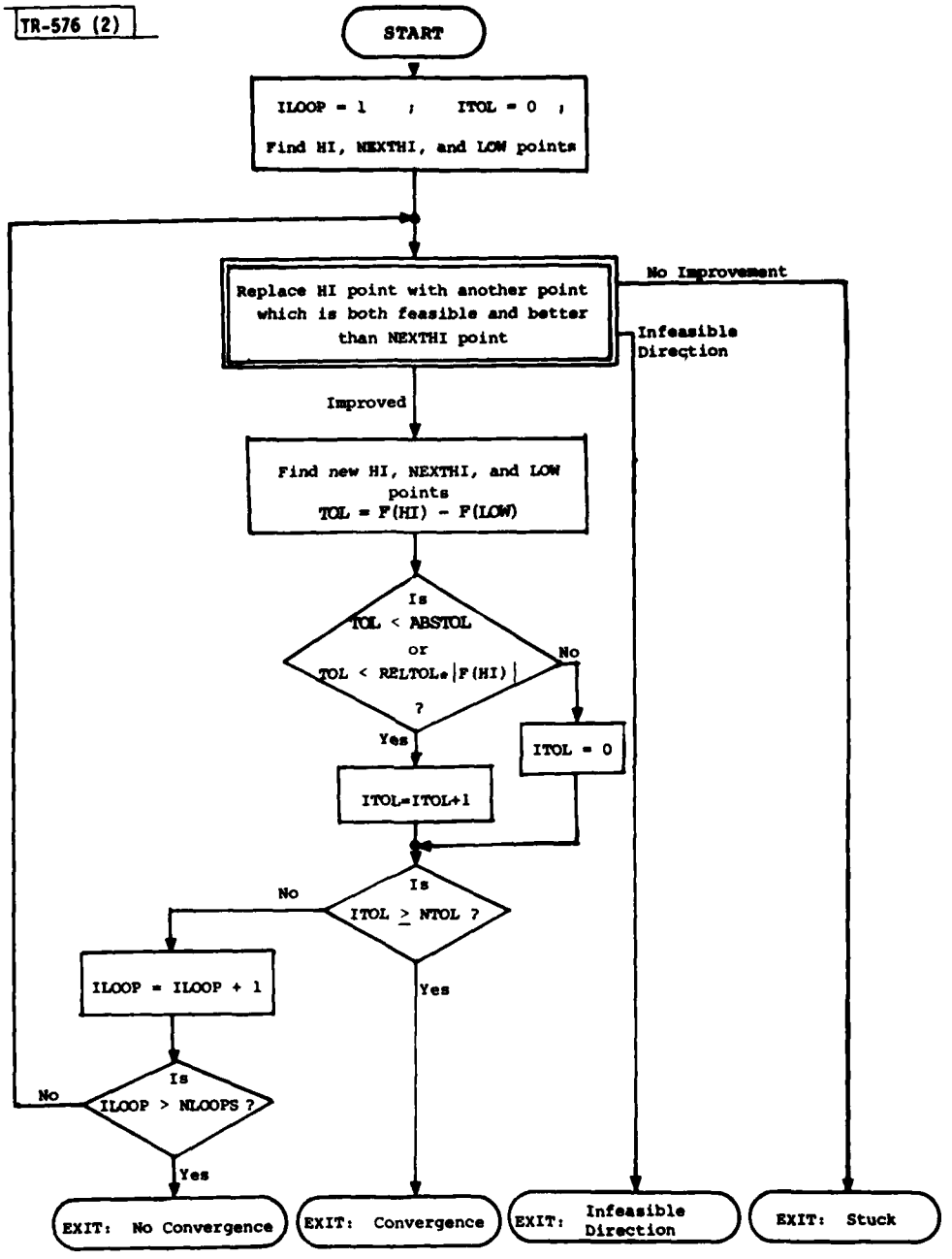


Fig. 2. Algorithm termination.

of these retractions fails to result in feasibility, we begin again with a new random point.

In the remaining circumstances we are effectively ignorant of both the geometry of Ω and the behavior of f. If so, we utilize option c) by making an initial random search of Ω and retaining the best KPTS points to form the initial complex. The only complication here is the number of random points (denoted NRANPT in the program) to utilize. Clearly we must have NRANPT $\geq$ KPTS, but also in practice we must confront the trade-off between the cost in terms of function evaluations of taking NRANPT large vs. the cost in terms of low probability of being close to the minimum if NRANPT is small.

One way to sensibly determine a value of NRANPT is to require a certain probability of placing at least k points inside a small neighborhood V of the global minimum of f in Ω. If the relative volume of V in Ω is α, then

$$\text{Prob}(\geq k \text{ points in V in N tries})$$

$$= 1 - \sum_{i=0}^{k-1} \binom{N}{i} \alpha^i (1-\alpha)^{N-i}$$

$$= 1 - (1-\alpha)^N \sum_{i=0}^{k-1} \left(\frac{\alpha}{1-\alpha}\right)^i \frac{1}{i!} \quad \frac{N!}{(N-i)!}$$

$$\geq 1 - (1-\alpha)^N \, N^{k-1} \exp\left(\frac{\alpha}{1-\alpha}\right),$$

so that to make this probability exceed β it suffices to choose N so large that

$$(1-\alpha)^N N^{k-1} \exp \frac{\alpha}{1-\alpha} \leq 1-\beta.$$

In the special case where k=1, the "exp" term can be dropped from the left hand side.

To illustrate, let us take V be to a cube of side length 2r centered at the global minimum. In general V need not exist if the minimum is too near the boundary of Ω. However, if there are no implicit constraints, or if we can be sure for other reasons that the minimum is well inside Ω, then it makes sense to discuss V. Now $\alpha=(2r)^n/\text{vol}(\Omega)$ and if k=1 we need to choose N so that

$$N \geq \frac{\log(1-\beta)}{\log(1-\alpha)} \geq \frac{\log(1-\beta)}{-\alpha} = \frac{-\log(1-\beta)}{(2r)^n} \text{vol}(\Omega).$$

The least such N are displayed in Fig. 3 for nominal values of β and n, with r set = 0.1 and $\text{vol}(\Omega)=1$. One can easily note a very prominent "curse of dimensionality" here. This concludes our discussion of the initialization step in the program.

The progress of the algorithm when applied to a particular problem can be monitored as desired through the use of four print options. These are set by assigning to a variable IPRINT values = 0,1,2,3. The 0 value supresses all printing; this might be appropriate when the optimization is serving as a subroutine to a larger program. The 1 value causes only the final answer (best point and least function value) and machine

β \ n	2	5	10
0.5	18	2166	6,769,016
0.9	58	7196	22,486,183
0.95	75	9362	29,255,198

Fig. 3. Least number of random samples to give probability β of at least one sample falling in cube of radius 0.1 within a unit n-volume.

execution time to be displayed. The 2 value results in display of the best initial complex point and corresponding function value followed by each new improved point, the corresponding lower function value, and the loop number w en this improvement occurred. Finally, the 3 value causes the entire initial complex to be displayed, followed by further information for every iteration, whether or not an improvement was achieved. Specifically, the new point and its function value are displayed along with the index of the point in the previous complex which it replaces, and the number of function evaluations required to obtain the new point.

It remains to discuss the continuation options. Recall that the program will terminate for one of three reasons: convergence criterion met, number of iterations loops exceeded, or prematurely, because of failure to construct a new complex. Use of the IPRINT =2 or 3 option will have enabled the user to gain an impression of the progress made and its rate since initialization. If this progress is satisfactory but the convergence criterion

is unmet, the user will probably wish to continue from the most recent complex. On the other hand, if the algorithm appears to be struggling, the user is also given the option of restarting around the best point discovered. That is, the best point is now denoted $x^{(0)}$ and initialization option b) (defined on p. 16) is implemented.

IV. PARAMETER AND SUBROUTINE SPECIFICATIONS

The (FORTRAN) program consists of a main program, seven general subroutines, and two user supplied subroutines. In greater detail these are described next.

SUBROUTINE BOX is called by the main program to set up the initial complex, call the basic iterative subroutine GOBOX, and print the final answer.

SUBROUTINE GOBOX is the basic iterative subroutine; it accepts an initial complex and iteratively constructs new ones until termination. As necessary it calls the special purpose subroutines MOVE, FEASBL, CENTER, and the function evaluator FUNC.

SUBROUTINE MOVE repeatedly defines a new point halfway between two given points until certain termination criteria are met.

SUBROUTINE FEASBL checks that a given vector belongs to sets Ω of the form (2). If not it attempts to reposition this vector so as to meet all constraints.

SUBROUTINE CENTER computes the centroid of a complex less one point.

SUBROUTINE GB0 defines an all random initial complex according to initialization option c).

SUBROUTINE GB1 defines a random complex to include one given feasible point according to initialization option b).

SUBROUTINE FUNC computes the value of a given function

at a given point in its domain.

SUBROUTINE IMPLCT defines the various implicit constraint functions g_j appearing in (2).

Program parameters and control variables are user supplied as needed through a NAMELIST specification statement. It assigns the name "GO" to refer to the following list of variables and array names...

Problem description:

NX: number of independent variables

NIC: number of implicit constraint functions g_j in (2)

MAXMIM: set = +1(-1) to maximize (minimize)

XMIN: array of lower bounds for independent variables

XMAX: array of upper bounds for independent variables

XIMIN: array of lower bounds on constraint functions

XIMAX: array of upper bounds on constraint functions

Basic iteration controls:

KPTSET: used to prescribe number of points in complex

N1CUTS: upper bound on number of halfway moves towards centroid

N2CUTS: upper bound on number of halfway moves away from centroid

STEP: step size of reflection in basic iteration (see (3))

NLOOPS: upper bound on number of basic iterations

NTOL: convergence parameter = number of consecutive iterations without sufficient progress.

ABSTOL: convergence parameter (see (4))

RELTOL: convergence parameter (see (5))

Initialization and run parameters:

IRUN: program operation control; usage defined in section V

IPRINT: output control; usage defined in section III

MAXRAN: bound on number of attempts to establish initial complex

NRANPT: total number of random draws from feasible set to define initial all-random complex

SEED: seed used in system uniform random number generator

XO: initial feasible point

XXO: initial feasible complex

Of these variables it is imperative that the user assign values to the first five and, if NIC > 0, to the next two also. Having done so the program will run using the default settings of the remaining parameters; in particular an all-random initial complex will be chosen.

In practice, the user will want to consider the appropriate values for the initialization and convergence parameters. According to the initialization option chosen, one of XO, XXO or NRANPT must be defined. The user will also probably need to assign relevant values to the convergence tolerance values ABSTOL and RELTOL, and the iteration bound NLOOPS. By default, ABSTOL=0.,RELTOL= 0.000001 and NLOOPS=500. Also, by default, STEP=1.5,

KPTS=IFIX (1.5*NX+0.5), N1CUTS=8, N2CUTS=16, and IPRINT=1. The use of the program control IRUN is described in the next section, along with several examples.

As already noted the STEP parameter should satisfy $1.0 < STEP \leq 2.0$. Ideally one would like STEP larger initially to make rapid progress towards the solution point and smaller as this point is approached. In general, larger values of STEP can result in infeasible reflected points and consequent loss of time in retreating towards the centroid, while small values of STEP result in slow progress and increased iterations. However, as the length of a reflection step is roughly proportional to the diameter of the associated complex, the step lengths automatically decrease as the algorithm progresses and the complexes shrink. Hence algorithm performance is relatively indifferent to moderate changes in STEP and STEP = 1.5 has proved to be a satisfactory default value.

Finally, there is the matter of "relevant values" for the tolerances ABSTOL and RELTOL. In general, only one of these needs be positive. If the user has a good idea of the order of magnitude of the optimal function value (as, for instance, in Example 4 of the next section), then ABSTOL should be set according to the final accuracy desired. If, on the other hand, the user has no reliable guide to the magnitude of the final function values, we suggest the default values of these two tolerance parameters. These cases are illustrated in Examples 5, 6 below where, in fact, even smaller values of RELTOL are employed.

V. PROGRAM OPERATION AND EXAMPLES

To utilize the program for the solution of a particular optimization problem of the form (1), (2), the user should do the following:

1) define the objective function f in SUBROUTINE FUNC, and the constraint functions g_j in SUBROUTINE IMPLCT. In the latter the notation XI(J) is used for g_j, with J=1, 2, ..., NIC;

2) alter the DIMENSION statement in the main program, if necessary, to accomodate values NX>10 or NIC>9. Compile and run, then:

3) input desired values of the relevant NAMELIST variables, finally assigning a value to the control IRUN from among the following options.

IRUN = 0: stop

1: begin with user specified complex XXO

2: begin with user specified feasible point XO

3: begin with all-random complex (specify NRANPT)

4: continue with further iterations from most recent complex

5: restart as in IRUN = 2 with XO set equal current best point

6: reset all NAMELIST variables to default values

7: list current value of all NAMELIST variables

We now illustrate the operation of the program and the results obtained on a variety of examples. The actual outputs reported below result from operation of a double precision FORTRAN

version of our method in CMS on an Amdahl 470/V7 mainframe computer. A listing of this version appears in the Appendix.

Example 1. Minimize $100(x_2-x_1^2)^2 + (1-x_1)^2$.

This is a famous test problem associated with the name of Rosenbrock. It is an unconstrained minimization; the objective function surface is notable for a narrow deep curving valley. The true solution is clearly 0 at (1,1). We implement the program by setting NX=2, NIC=0, XMIN=2*-2., XMAX=2*2. (thereby constraining (x_1,x_2) to the cube defined by $|x_i| \leq 2$), and MAXMIN = -1. We report in Fig. 4 below the results of three sets of 10 runs with the indicated values of ABSTOL, RELTOL, and STEP. (In all the examples the default value STEP = 1.5 is used unless otherwise noted.) In all cases the conventional starting value XO = (-1.2, 1.0) was used (IRUN = 2). For this example, and in general when we can reasonably guess at the magnitude of the optimal function value, we should set RELTOL = 0.0 and choose ABSTOL according to the desired accuracy. We did so in the second set of runs here, desiring that F be reduced below 10^{-10}. The third set is the same except for STEP = 1.3 (the original Box recommendation); in this case we note a slightly poorer and more erratic result.

	Function Evaluations	Final Function Value
ABSTOL = 0.0 RELTOL = 0.000001	916 (323)	1.029 D-24 (2.106 D-24)
ABSTOL = 5.0D-11 RELTOL = 0.0	360 (53)	5.729 D-11 (10.780 D-11)
ABSTOL = 5.0D-11 RELTOL = 0.0 STEP = 1.3	317 (63)	6.773 D-09 (18.333)D-09

Fig. 4. Means and (standard errors) for the Rosenbrock function minimization.

We note that since feasible points are readily available in this example, the pure random start, being wasteful of function evaluations, is avoided.

Example 2. Maximixe $x_1^2+4x_1x_2+7x_2^2$ over the set $\Omega=\{(x_1,x_2): 0\leq x_i\leq 1, -1\leq x_1+2x_2\leq 1, -1\leq 3x_1-4x_2\leq 1\}$. This is a concave quadratic program over the pentagonal region Ω with exact solution 1.48 at the vertex (0.2, 0.4). We set NX=NIC=2, XMIN=2*0.1, XMAX=2*1.1, XIMIN=2*-1.1, XIMAX=2*1.1, ABSTOL=RELTOL=.000001, and made ten runs starting from XO=(0.3, 0.2). The results were

	function evaluations	function value	solution point
mean (standard error)	54 (10)	1.478883 (0.000012)	(.200004, .399998) ((.000010, .000005)).

Example 3. Minimize $4x_1^2-2.1x_1^4+x_1^6/3+x_1x_2-4x_2^2+4x_2^4$ over the rectangle $\Omega = \{(x_1, x_2): |x_1|\leq 2.5, |x_2|\leq 1.5\}$. This is another famous test problem, the objective function being known as the "six hump camel back function" [17]. It is a challenge for local gradient-utilizing methods because of the presence of an assortment of stationary points: 6 local minima, 2 local maxima, and 7 saddle points. Published numerical tests of other global optimization procedures [17] give a global minimum value of -1.0316 at the points (0.08984, -0.71266) and (0.71266, -0.08984), at a cost of 72-800 function evaluations. We set ABSTOL=RELTOL=0.0001 and made

5 runs each with KPTS=3, KPTS=4, starting in all cases at XO=(0,0) (IRUN=2). The results were

	function evaluations	function value
KPTS=3	132 (21)	-1.031626 (0.000001)
KPTS=4	130 (14)	-1.031617 (0.000012)

We note about the same amount of work involved with the larger complex, but also a slightly greater variability in the final answer.

Example 4. Solve the nonlinear system of equations

$$2x_1^3x_2 = x_2^3$$

$$6x_1 - x_2^2 + x_2 = 0.$$

By sketching the corresponding pair of curves in the (x_1,x_2) - plane, one can easily verify a pair of exact solutions at (0,0) and (2,4) and one other solution near (1.5, - 3.0). To determine this third solution exactly we try to minimize the sum of the squared residuals; that is, minimize $(2x_1^3x_2-x_2^3)^2 + (6x_1-x_2^2+x_2)^2$. The least value is of course 0, and the solution point will be our unknown third solution. We set $\Omega=\{(x_1, x_2): 1\leq x_1\leq 2, -4\leq x_2\leq -2\}$,

XO=(1.5, - 3.0), and ABSTOL = 1.0D-13, RELTOL = 0.0, (attempting to obtain six or seven significant figures). The results of five runs were

	function evaluations	solution point
mean	291	(1.464352, -2.506013)
(standard error)	(24)	(0.000000, 0.000000)

In all cases the objective function was reduced to less than 5.0D-12, with an average final value = 8.85D-13.

Example 5. Minimize $x_1^2+x_2^2+x_3^2$ over the region $\Omega=\{(x_1, x_2, x_3): |x_i|\leq 10, 3\leq x_1x_2x_3, 3\leq x_1+x_2-x_3\}$. This problem is unusual because Ω is disconnected: it consists of three components each of equal distance from the origin. Hence there are three distinct solution points, each of which is a boundary point. These points can be obtained via the Kuhn-Tucker conditions and analysis of the possible sign patterns of the coordinates, viz., (+,+,+), (+,-,-), and (-,+,-). Thus the first point must obey the conditions

$$x_1 = x_2$$
$$x_1x_2x_3 = 3$$
$$x_1+x_2-x_3 = 3.$$

This leads to the cubic equations $x^2(2x-3)=3$ for x_1; its solution = 1.910820 = $x_2 \triangleq a$. Therefore, $x_3=2x_1-3$ = 0.821640$\triangleq$b. By symmetry, the other solution points are (a, -b, -a) and (-b, a, -a), and the

common minimum function value = 7.977559.

This example is meant to typify the situation wherein we are ignorant of the geometry of the feasible set Ω. In such cases we must resort to the pure random start option (IRUN=3). Setting NRANPT = 500, ABSTOL = 0., and RELTOL = 1.0D-08, the program easily reveals the three distinct solution points. The results of ten runs were

	function evaluations	function value
mean (standard error)	1043 (253)	7.977583 (0.000046)

Eight of the ten runs actually resulted in function values $\leq$7.977560.

Example 6. Minimize $100(x_2-x_1^2)^2 + (1-x_1)^2 + 90(x_4-x_3^2)^2 + (1-x_3)^2 + 10.1[(x_2-1)^2 + (x_4-1)^2] + 19.8(x_2-1)(x_4-1)$, over the four-dimensional cube $\Omega=\{(x_1,x_2,x_3,x_4): |x_i| \leq 10\}$. This is another test problem associated with the name of Wood [6, p. 403]; it is designed to have a non-optimal stationary point with a corresponding function value of approximately 8.0 that can cause premature convergence. The optimal solution is clearly 0 at (1,1,1,1). A starting value XO=(-3,-1,-3,-1) is suggested. Our program handled this problem easily with no premature stalls. The results of ten runs with ABSTOL=0., RELTOL=1.0D-10 (the value of the objective function was increased by 1) uniformly yielded function values

< 1.0000000005. with an average of 1145 (198) function values.

Example 7. Our final example is a nonlinear regression taken from [24]. The problem is to model the resistance (R) of a thermistor as a function of temperature (T) via the model

$$R = a \exp\left(\frac{b}{T + c}\right).$$

Specifically we want to assign values to the parameters a,b,c so as to achieve the best fit to the data

T	R
50	34,780
55	28,610
60	23,650
65	19,630
70	16,370
75	13,720
80	11,540
85	9,744
90	8,261
95	7,030
100	6,005
105	5,147
110	4,427
115	3,820
120	3,307
125	2,872

We do so by minimizing the mean squared residual

$$\left(\sum_{j=1}^{16}\left(R_j - a \exp\left(\frac{b}{T_j+c}\right)\right)^2\right)^{1/2}.$$

The source cited reports that this problem caused considerable difficulity to the algorithms being tested. In general we note that the special structure of nonlinear least squares problems can be utilized in the design of specialized algorithms such as the methods of Gauss-Newton or Levenberg-Marquardt [25]. For the latest in such algorithms, see [26]. These may certainly be expected to be far more efficient than our present all-purpose method. Nevertheless, it is interesting to test our method on such a problem, recalling particularly the discussion of the computer time vs. analyst time trade-off.

The suggested initial values are XO=(0.2, 4000, 250.), the corresponding function value being 41,153. We set STEP=2. and ABSTOL = .0001. = RELTOL . We also introduce the artifical constraint $g_1(a,b,c) \triangleq b/(50+c) \leq 70$ in order to avoid overflow problems in the exponential. The outcomes of ten runs were classified as success or failure according as convergence was or was not achieved with at most five restarts (IRUN=5). The results were

	number	function evaluations	function value	CPU time (sec.)
success	6	6480 (1250)	9.3788 (.0010)	2.94 (.58)
failure	4	3031 (1340)	267.74 (90)	1.24 (.70)

The mean solution point for the six successes was

(.005612, 6181.00, 345.210)

with standard errors (.000006, 0.85, 0.030); this may be compared with the optimal solution reported in [24]:

(.005609, 6181., 345.2).

VI. CONCLUSIONS

The numerical practice of nonlinear multivariate optimization is an important, difficult, and much studied subject. Numerous algorithms have been proposed, some exotic, some effective, most requiring special hypotheses on the objective and constraint functions. We have presented here an extremely simple and robust procedure for handling the most general nonlinear inequality constrained optimization, and demonstrated its operation and effectiveness on a variety of test problems of low dimension. The method utilizes the basic Box complex iteration with modifications to avoid traps. The computer implementation provides a variety of initialization and continuation features and is designed for interactive use. While problems of a special nature can be solved more efficiently with specially formulated algorithms, in a certain practical sense our method is reasonably competitive with these, and moreover constitutes a safe reliable procedure for use on problems with little or no exploitable structure. Examples of the latter, on which our method has been successfully applied, include minimizing a discontinuous cost function subject to a communications network reliability constraint, and minimizing a two-stage survival probability function to achieve an optimal attack against a given layered ballistic missile defense system.

REFERENCES

1. T. Arthamari and Y. Dodge, Mathematical Programming in Statistics (Whiley, New York, 1981).

2. M. Avriel, Nonlinear Programing (Prentice-Hall, Englewood Cliffs, New Jersey, 1976).

3. M. Bazaraa and C. Shetty, Nonlinear Programming (Wiley, New York, 1979).

4. P. Gill and W. Murray, ed's., Numerical Methods for Constrained Optimization (Academic Press, New York, 1974).

5. M. Hestenes, Optimization Theory (Wiley, New York,1975).

6. D. Himmelblau, Applied Nonlinear Programming (McGraw-Hill, New York,1972).

7. G. Walsh, Methods of Optimization (Wiley, New York, 1975).

8. D. Wismer and R. Chattergy, Introduction to Nonlinear Optimization (North-Holland, New York, 1978).

9. H. Kuhn and A. Tucker, "Nonlinear Porgramming", in Proc. 2nd Berkeley Symp. on Mathematical Statistics and Probability, J. Neyman, ed., Univ. of California Press, 1951.

10. G. Zoutendijk, Mathematical Programming Methods (North-Holland, New York, 1976).

11. J. Rosen, "The Gradient Projection Method for Nonlinear Programming, Part I. Linear Constraints", SIAM J. Applied Math. 8, 181, (1960).

12. R. Griffith and R. Stewart, A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems", Management Science 7, 379, (1961).

13. R. Fletcher, "An Algorithm for Solving Linearly Constrained Optimization Problems", Math. Prog. 2, 133, (1972).

14. E. Levitan and B. Polyak, "Constrained Minimization Methods", USSR Comp. Math. and Math. Physics 6, 1, (1966).

15. J. May, "Solving Nonlinear Programs Without Using Analytic Derivatives", Operations Research 27, 457, (1979).

16. L. Dixon and G. Szegö, ed's., Towards Global Optimization (North-Holland, New York,1975).

17. L. Dixon and G. Szegö, ed's., Towards Global Optimization (North-Holland, New York, 1978).

18. F. Archetti and G. Szegö, "Global Optimization Algorithms", in Nonlinear Optimization, L. Dixon, E. Spedicato and G. Szegö, ed's. (Birkhäuser, 1980).

19. A. Waren and L. Lasdon, "The Status of Nonlinear Programming Software", Operations Research 27, 431, (1979).

20. J. Kuester and J. Mize, Optimization Techniques with FORTRAN (McGraw-Hill, New York, 1973).

21. J. Lyness, "Has Numerical Differentiation A Future?", in Proc. 7th Manitoba Conf. on Numerical Mathematics and Computing, D. McCarthy and H. Williams, ed's., Utilitas Mathematica Publishing (1978).

22. R. Steplemand and N. Winarsky, "Adaptive Numerical Differentiation", Math. Comp. 33, 1257 (1979).

23. H. Mosteller, "Heuristic Direct-Search Minimization", IEEE Trans. Auto. Control 23, 493 (1978).

24. M. Box, "A New Method of Constrained Optimization and a Comparsion With Other Methods", Computer J. 8, 42 (1965).

25. R. Meyer, "Theoretical and Computational Aspects of Nonlinear Regression", in Nonlinear Programming, J. Rosen, O. Mangassarian and K. Ritter, ed's. (Academic Press, New York, 1970).

26. A. Ruhe and P. Wedin, "Algorithms for Separable Nonlinear Least Squares Problems", SIAM Review 22, 318 (1980).

Appendix

This appendix contains the complete FORTRAN listing of the optimization program described above. The data for the function and implicit constraint portions, SUBROUTINES FUNC and IMPLCT respectively, correspond to Example 7 in Section V.

```
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION XX(10,15),XSTAR(10),XMIN(10),XMAX(10),X0(10)
      DIMENSION XIMIN(9),XIMAX(9),XI(9),FF(15),XX0(10,15),X(10)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *               ,IPRINT,MAXMIN,SEED,NRANPT,MAXRAN
      REAL*8 LXMIN,LXMAX,LXIMIN,LXIMAX,LX0,LXX0,LBLANK
      NAMELIST /GO/ NX,NIC,MAXMIN,XMIN,XMAX,XIMIN,XIMAX,
     *              KPTSET,N1CUTS,N2CUTS,STEP,NLOOPS,NTOL,ABSTOL,RELTOL,
     *              IRUN,IPRINT,NRANPT,MAXRAN,SEED,X0,XX0
      DATA LXMIN,LXMAX   /8H XMIN   ,8H XMAX   /
      DATA LXIMIN,LXIMAX /8H XIMIN  ,8H XIMAX  /
      DATA LX0,LXX0,LBLANK /8H   X0   ,8H  XX0   ,8H        /
C
C  PROGRAM NEWBOX USES SUBROUTINE BOX TO SOLVE THE PROBLEM
C        MAXIMIZE    MAXMIN*FUNC(X)
C        SUBJECT TO  XMIN(I)  .LE. X(I)  .LE. XMAX(I)    ,I=1,...,NX
C                    XIMIN(J) .LE. XI(J) .LE. XIMAX(J)   ,J=1,...,NIC
C  WHERE XI(J) IS THE VALUE OF THE JTH IMPLICIT CONSTRAINT FUNCTION
C  EVALUATED AT THE POINT X, AS COMPUTED IN SUBROUTINE IMPLCT.
C
C  THE USER MUST SUPPLY SUBROUTINE FUNC(X,NX,F) WHICH CALCULATES F,
C  THE FUNCTIONAL VALUE OF A POINT X, AND A SUBROUTINE
C  IMPLCT(X,NX,XI,NIC) WHICH CALCULATES XI, THE NIC IMPLICIT
C  CONSTRAINT VALUES CORRESPONDING TO THE POINT X.
C
C
C  SET PARAMETERS TO THEIR DEFAULT VALUES.
C
  100 KPTSET=0
      N1CUTS=8
      N2CUTS=16
      STEP=1.5
      NLOOPS=500
      NTOL=5
      ABSTOL=0.
      RELTOL=.000001
      IRUN=3
      IPRINT=1
      NRANPT=0
      MAXRAN=100
      SEED=184287807
C
```

```
C  READ NAMELIST &GO DATA DEFINING PARAMETERS FOR NEXT RUN
C
  200 WRITE(6,9)
    9 FORMAT('1ENTER NAMELIST &GO INPUT (IRUN=7 LISTS ALL)')
      READ(5,GO)
      IF(IRUN.LE.0) GO TO 1000
      IF(IRUN-6) 300,100,400
C
C  CALL BOX TO OPTIMIZE FOR THE CURRENT PARAMETERS
C
  300 KPTS=NX+(NX+1)/2
      IF(KPTSET.GE.2) KPTS=KPTSET
      CALL BOX(XX,NX,KPTS,NIC,XMIN,XMAX,XIMIN,XIMAX
     *        ,FF,XI,XSTAR,FSTAR,X,X0,XX0,IRUN)
      GO TO 200
C
C  PRINT THE CURRENT VALUES OF THE PARAMETERS
C
  400 WRITE(6,1) NX,NIC,MAXMIN
    1 FORMAT('0CURRENT VALUES OF PARAMETERS ARE:',/,
     *       '0',10X,'PROBLEM DESCRIPTION',/,
     *       ' NX,NIC,MAXMIN = ',3I5)
      WRITE(6,2) LXMIN,(XMIN(I),I=1,NX)
      WRITE(6,2) LXMAX,(XMAX(I),I=1,NX)
      WRITE(6,2) LXIMIN,(XIMIN(I),I=1,NIC)
      WRITE(6,2) LXIMAX,(XIMAX(I),I=1,NIC)
    2 FORMAT(A8,'= ',1P10E12.4,/,(10X,1P10E12.4))
      WRITE(6,3) KPTSET,N1CUTS,N2CUTS,STEP,NLOOPS,NTOL,ABSTOL,RELTOL
    3 FORMAT('0',10X,'BOX PARAMETERS',/,
     *       ' KPTSET,N1CUTS,N2CUTS,STEP = ',3I5,F12.6,/,
     *       ' NLOOPS,NTOL,ABSTOL,RELTOL = ',2I5,1P2E12.3)
      WRITE(6,4) IRUN,IPRINT,NRANPT,MAXRAN,SEED
    4 FORMAT('0',10X,'RUN INITIALIZATION PARAMETERS',/,
     *       'IRUN,IPRINT,NRANPT,MAXRAN,SEED = ',2I4,2I7,3X,Z16)
      WRITE(6,2) LX0,(X0(I),I=1,NX)
      WRITE(6,2) LXX0,(XX0(I,1),I=1,NX)
      DO 440 J=2,KPTS
  440 WRITE(6,2) LBLANK,(XX0(I,J),I=1,NX)
      GO TO 200
C
 1000 STOP
      END
```

```
      SUBROUTINE BOX(XX,NX,KPTS,NIC,XMIN,XMAX,XIMIN,XIMAX,FF,XI
     *               ,XSTAR,FSTAR,X,X0,XX0,IRUN)
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION XX(NX,KPTS),XMIN(NX),XMAX(NX),XIMIN(NIC),XIMAX(NIC)
      DIMENSION FF(KPTS),XI(NIC),XSTAR(NX)
      DIMENSION X(NX),X0(NX),XX0(NX,KPTS)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *                ,IPRINT,MAXMIN,SEED,NRANPT
      REAL*8 DDATE,DTIME
C
C  BOX SETS UP THE INITIAL COMPLEX THEN CALLS GOBOX TO FIND THE
C  OPTIMAL NX-VECTOR XSTAR AND ITS FUNCTIONAL VALUE FSTAR.
C
      GO TO (100,200,300,400,500),IRUN
C
C  IRUN=1 . . . . SET INITIAL XX TO USER-DEFINED XX0
C
  100 WRITE(6,1) KPTS,STEP
    1 FORMAT('0ENTERING BOX WITH DEFAULT XX',/
     *       ,'       KPTS,STEP = ',I5,F10.6)
      DO 150 J=1,KPTS
         DO 140 I=1,NX
  140    XX(I,J)=XX0(I,J)
         CALL FUNC(XX(1,J),NX,FF(J))
  150    CONTINUE
      NFUNC=KPTS
      GO TO 900
C
C  IRUN=2 . . . . SET INITIAL XX RANDOMLY AROUND (FEASIBLE) X0
C
  200 WRITE(6,2) SEED,KPTS,STEP,X0
    2 FORMAT('0ENTERING BOX WITH RANDOM XX, SEED = ',D13.5,/
     *       ,'      KPTS,STEP = ',I5,F10.6,/
     *       ,'      X0 =',1P9E12.4,/,(9X,1P9E12.4))
      CALL GB1(XX,NX,KPTS,X0,FF,XMIN,XMAX,XIMIN,XIMAX,XI
     *         ,NIC,IOK)
      NFUNC=KPTS
      IF(IOK.GT.0) GO TO 900
      GO TO 1000
C
C  IRUN=3 . . . . SET INITIAL XX RANDOMLY IN FEASIBLE REGION
C
  300 WRITE(6,3) SEED,KPTS,NRANPT,STEP
    3 FORMAT('0ENTERING BOX WITH ALL RANDOM XX, SEED = ',D13.5,/
     *       ,'       KPTS,NRANPT,STEP = ',2I5,F10.6)
      CALL GB0(XX,NX,KPTS,FF,XMIN,XMAX,XIMIN,XIMAX
     *         ,XI,NIC,X,IOK)
      NFUNC=MAX0(KPTS,NRANPT)
      IF(IOK.GT.0) GO TO 900
      GO TO 1000
C
```

```
C   IRUN=4 . . . . CONTINUE WITH XX AND FF AS LEFT BY PREVIOUS RUN
C
  400 WRITE(6,4) KPTS,NLOOPS,STEP
    4 FORMAT('0CONTINUING BOX WITH SAME XX',/
     *       ,'       KPTS,NLOOPS,STEP = ',2I5,F10.6)
      NFUNC=0
      GO TO 900
C
C   IRUN=5 . . . . SET INITIAL XX RANDOMLY AROUND LAST XSTAR
C
  500 WRITE(6,6) SEED,KPTS,STEP,XSTAR
    6 FORMAT('0RESTARTING BOX AROUND XSTAR, SEED =',D15.5,/
     *       ,'      KPTS,STEP =',I5,F10.6,/
     *       ,'      XSTAR =',1P9E12.4,/,(12X,1P9E12.4))
      CALL GB1(XX,NX,KPTS,XSTAR,FF,XMIN,XMAX,XIMIN,XIMAX,XI
     *         ,NIC,IOK)
      NFUNC=KPTS
      IF(IOK.GT.0) GO TO 900
      GO TO 1000
C
C   NOW CALL BOX WITH XX AND FF TO FIND OPTIMUM XSTAR AND FSTAR
C
  900 CALL TIMES(DDATE,DTIME,IVCPU1,ITCPU)
      CALL GOBOX(XX,NX,KPTS,NIC,XMIN,XMAX,XIMIN,XIMAX,FF,XI
     *          ,XSTAR,FSTAR,MLOOPS,NFUNC)
      FSTAR=MAXMIN*FSTAR
      CALL TIMES(DDATE,DTIME,IVCPU2,ITCPU)
      TIME=.000013*(IVCPU2-IVCPU1)
      IF(IPRINT.GT.0) WRITE(6,5) MLOOPS,NFUNC,TIME,FSTAR,XSTAR
    5 FORMAT('0',I5,' LOOPS,',I6,' FUNCS, AND',F7.3,' SECS LATER BOX'
     *,' FINDS FSTAR AND XSTAR OF',1PE16.9,/,(1P9E14.6))
C
 1000 RETURN
      END
```

```
      SUBROUTINE GOBOX(XX,NX,KPTS,NIC,XMIN,XMAX,XIMIN,XIMAX
     *                 ,FF,XI,XC,FSTAR,MLOOPS,NFUNC)
      IMPLICIT REAL*8 (A-H, O-Z)
      DIMENSION XX(NX,KPTS),XMIN(NX),XMAX(NX),XIMIN(NIC),XIMAX(NIC)
      DIMENSION FF(KPTS),XI(NIC),XC(NX)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *                ,IPRINT,MAXMIN
      DATA IO,I1 / 0,1 /
C
C  GIVEN AN INITIAL COMPLEX OF KPTS NX-VECTORS IN XX, AND KPTS
C  FUNCTION EVALUATIONS IN FF, GOBOX WILL TRY TO FIND THE POINT
C  WHICH MAXIMIZES FUNC ON THE FEASIBLE REGION.  THIS POINT IS
C  RETURNED IN ARRAY XC, AND ITS VALUE IN FSTAR.
C
C  FIND HI AND LO POINTS IN THE COMPLEX
C
      JLO=1
      JHI=1
      DO 40 J=2,KPTS
         IF(FF(J).LT.FF(JLO)) JLO=J
         IF(FF(J).GT.FF(JHI)) JHI=J
   40    CONTINUE
      ITOL=0
      IF(IPRINT-2) 100,50,60
   50 F=MAXMIN*FF(JHI)
      WRITE (6,2) IO,I1,JHI,F,(XX(I,JHI),I=1,NX)
      GO TO 100
   60 DO 70 J=1,KPTS
         F=MAXMIN*FF(J)
         IF(J.NE.JHI) WRITE(6,1) IO,I1,J,F,(XX(I,J),I=1,NX)
         IF(J.EQ.JHI) WRITE(6,2) IO,I1,J,F,(XX(I,J),I=1,NX)
    1    FORMAT('   LP,NF,J,F,X=',3I3,1PE12.4,1X,9E11.3,/
     *          ,(37X,1P9E11.3))
    2    FORMAT(' **LP,NF,J,F,X=',3I3,1PE12.4,1X,9E11.3,/
     *          ,(37X,1P9E11.3))
   70    CONTINUE
C
C  MAKE UP TO NLOOPS ITERATIONS, EACH TIME IMPROVING THE LOWEST VALUE
C
  100 DO 500 ILOOP=1,NLOOPS
C
C  LOCATE NEXT LOWEST POINT
C
         NORMLC=1
         IFUNC=NFUNC
         NEXTLO=JHI
         DO 140 J=1,KPTS
            IF(J.NE.JLO .AND. FF(J).LT.FF(NEXTLO)) NEXTLO=J
  140       CONTINUE
C
```

```
C  FIND CENTROID AND CHECK ITS FEASIBILITY.
C
      CALL CENTER(XX,NX,KPTS,JLO,XC)
      CALL FEASBL(XC,NX,XC,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *           ,NMOVES,0)
      IF(NMOVES.GT.0) GO TO 200
C
C  FOR FEASIBLE CENTROID, LOOK FOR NEW POINT BY MOVING FROM LO POINT
C  THRU CENTER BY FACTOR OF STEP, THEN MOVE TOWARDS CENTER IF NECESSARY
C  TO GET A POINT BOTH FEASIBLE AND BETTER THAN NEXTLO POINT.
C
      DO 150 I=1,NX
  150 XX(I,JLO)=XX(I,JLO)+(1.+STEP)*(XC(I)-XX(I,JLO))
      CALL MOVE(XX(1,JLO),NX,XC,NMOVES,N1CUTS,FF(JLO),FF(NEXTLO)
     *         ,NFUNC,XMIN,XMAX,XIMIN,XIMAX,XI,NIC)
      IF(NMOVES.LE.N1CUTS) GO TO 400
C
C  IF ABOVE DIDN'T WORK, OR IF CENTER NOT FEASIBLE, LOOK FOR NEW
C  POINT BY MOVING FROM CENTER TOWARDS HI POINT.
C
  200 DO 250 I=1,NX
  250 XX(I,JLO)=XC(I)
      NORMLC=-1
      CALL MOVE(XX(1,JLO),NX,XX(1,JHI),NMOVES,N2CUTS
     *         ,FF(JLO),FF(NEXTLO),NFUNC
     *         ,XMIN,XMAX,XIMIN,XIMAX,XI,NIC)
      IF(NMOVES.LE.N2CUTS) GO TO 400
C
C  IF THIS DIDN'T WORK, TEST ON BOTH SIDES OF HI POINT ALONG LINE
C  THRU CENTER FOR FEASIBILITY AND IMPROVEMENT.  IF BOTH ARE INFEAS-
C  IBLE QUIT WITH INFEASIBLE DIRECTION AT 800; IF ONE IS BOTH
C  FEASIBLE AND BETTER, CONTINUE WITH IT; OTHERWISE QUIT WITH NO
C  IMPROVEMENT AT 700
C
      CALL FEASBL(XX(1,JLO),NX,XX(1,JLO),XMIN,XMAX,XIMIN,XIMAX
     *           ,XI,NIC,NMOVES,0)
      IF(NMOVES.LE.0) GO TO 700
      DO 260 I=1,NX
  260 XX(I,JLO)=XX(I,JHI)+(XX(I,JHI)-XX(I,JLO))
      CALL FEASBL(XX(1,JLO),NX,XX(1,JLO),XMIN,XMAX,XIMIN,XIMAX
     *           ,XI,NIC,NMOVES,0)
      IF(NMOVES.GT.0) GO TO 800
      CALL FUNC(XX(1,JLO),NX,FF(JLO))
      NFUNC=NFUNC+1
      IF(FF(JLO).GT.FF(NEXTLO)) GO TO 400
C
```

```
C  NOW THAT WE HAVE IMPROVED THE LOW X, TEST FOR COMPLETION AND
C  LOOP AROUND AGAIN
C
  400     MFUNC=NORMLC*(NFUNC-IFUNC)
          F=MAXMIN*FF(JLO)
          IF(FF(JLO).LE.FF(JHI)) GO TO 440
          IF(IPRINT.GE.2) WRITE(6,2) ILOOP,MFUNC,JLO,F
     *                                ,(XX(I,JLO),I=1,NX)
          JHI=JLO
          GO TO 450
  440     IF(IPRINT.GE.3) WRITE(6,1) ILOOP,MFUNC,JLO,F
     *                                ,(XX(I,JLO),I=1,NX)
  450     JLO=NEXTLO
          ITOL=ITOL+1
          ATOL=DABS(FF(JHI)-FF(JLO))
          IF(ATOL.LE.ABSTOL) GO TO 480
          IF(ATOL.GT.RELTOL*DABS(FF(JHI))) ITOL=0
  480     IF(ITOL.GE.NTOL) GO TO 600
  500     CONTINUE
C
C                              NO CONVERGENCE
       MLOOPS=NLOOPS+1
       GO TO 900
C                              CONVERGENCE
  600  MLOOPS=ILOOP
       GO TO 900
C                              UNABLE TO IMPROVE
  700  MLOOPS=-ILOOP
       GO TO 900
C                              INFEASIBLE DIRECTION
  800  MLOOPS=-NLOOPS-1
C
  900  DO 950 I=1,NX
  950  XC(I)=XX(I,JHI)
       FSTAR=FF(JHI)
C
       RETURN
       END
```

```
      SUBROUTINE FEASBL(X,NX,XC,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *                  ,NMOVES,MAXMOV)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(NX),XC(NX),XMIN(NX),XMAX(NX)
      DIMENSION XIMIN(NIC),XIMAX(NIC),XI(NIC)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *               ,IPRINT,MAXMIN
C
C  TRIES TO MAKE VECTOR X FEASIBLE BY (IF NECESSARY) MOVING X
C  TOWARDS GIVEN FEASIBLE VECTOR XC UP TO MAXMOV TIMES.
C
C  CHECK AGAINST EXPLICIT CONSTRAINTS
C
      NMOVES=0
  100 DO 200 I=1,NX
         IF(X(I).LT.XMIN(I) .OR. X(I).GT.XMAX(I)) GO TO 400
  200    CONTINUE
C
C  CHECK AGAINST IMPLICIT CONSTRAINTS
C
      IF(NIC.LE.0) RETURN
      CALL IMPLCT(X,NX,XI,NIC)
      DO 300 I=1,NIC
         IF(XI(I).LT.XIMIN(I) .OR. XI(I).GT.XIMAX(I)) GO TO 400
  300    CONTINUE
      RETURN
C
C  INCREMENT NMOVES AND IF .LE. MAXMOV, MOVE X TOWARDS CENTROID.
C
  400 NMOVES=NMOVES+1
      IF(NMOVES.GT.MAXMOV) RETURN
      DO 500 I=1,NX
  500 X(I)=0.5*(X(I)+XC(I))
      GO TO 100
C
      END
```

```
      SUBROUTINE MOVE(X,NX,XT,NMOVES,MAXMOV,F,FMIN,NFUNC
     *                ,XMIN,XMAX,XIMIN,XIMAX,XI,NIC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(NX),XT(NX),XMIN(NX),XMAX(NX),XIMIN(NIC)
      DIMENSION XIMAX(NIC),XI(NIC)
C
C  TRIES TO VERIFY THAT VECTOR X IS BOTH FEASIBLE AND HAS VALUE
C  .GT. FMIN, IF NECESSARY BY MOVING X TOWARDS THE VECTOR XT UP
C  TO MAXMOV TIMES.
C
      NMOVES=0
  100 CALL FEASBL(X,NX,XT,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *           ,IMOVES,MAXMOV-NMOVES)
      NMOVES=NMOVES+IMOVES
      IF(NMOVES.GT.MAXMOV) RETURN
      CALL FUNC(X,NX,F)
      NFUNC=NFUNC+1
      IF(F.GT.FMIN) RETURN
      DO 200 I=1,NX
  200 X(I)=0.5*(X(I)+XT(I))
      NMOVES=NMOVES+1
      IF(NMOVES.LE.MAXMOV) GO TO 100
C
      END



      SUBROUTINE CENTER(XX,NX,KPTS,NOTJ,XC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XX(NX,KPTS),XC(NX)
C
C  COMPUTES THE NX-VECTOR XC AS THE CENTROID OF THE KPTS NX-VECTORS
C  IN XX LESS THE ONE INDEXED BY NOTJ.
C
      DIVKM1=1.0/(KPTS-1)
      DO 200 I=1,NX
         XSUM=0.
         DO 100 J=1,KPTS
  100    XSUM=XSUM+XX(I,J)
         XC(I)=(XSUM-XX(I,NOTJ))*DIVKM1
  200    CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE GBO(XX,NX,KPTS,FF,XMIN,XMAX,XIMIN,XIMAX
     *              ,XI,NIC,X,IOK)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XX(NX,KPTS),FF(KPTS),XMIN(NX),XMAX(NX)
      DIMENSION XIMIN(NIC),XIMAX(NIC),XI(NIC),X(NX)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *               ,IPRINT,MAXMIN,SEED,NRANPT,MAXRAN
C
C  CREATES A RANDOM INITIAL COMPLEX, XX, BY CHOOSING THE BEST KPTS
C  POINTS OUT OF A COLLECTION OF MAX(KPTS,NRANPT) RANDOM VECTORS,
C  EACH UNIFORMLY DISTRIBUTED OVER THE FEASIBLE REGION.
C
C  FILL XX WITH KPTS FEASIBLE RANDOM VECTORS
C
      NRANX=0
      MXRANX=MAXRAN*MAX0(KPTS,NRANPT)
      DO 300 J=1,KPTS
  100    DO 200 I=1,NX
  200    XX(I,J)=XMIN(I)+RAN2(SEED)*(XMAX(I)-XMIN(I))
         NRANX=NRANX+1
         IF(NRANX.GT.MXRANX) GO TO 1100
         CALL FEASBL(XX(1,J),NX,XX(1,J),XMIN,XMAX,XIMIN,XIMAX,XI
     *              ,NIC,NMOVES,0)
         IF(NMOVES.GT.0) GO TO 100
         CALL FUNC(XX(1,J),NX,FF(J))
  300    CONTINUE
C
      K1=KPTS+1
      IF(K1.GT.NRANPT) GO TO 1000
      JMINF=1
      DO 500 J=2,KPTS
  500 IF(FF(J).LT.FF(JMINF)) JMINF=J
C
```

```
C  CHOSE (NRANPT-KPTS) RANDOM VECTORS, ALWAYS KEEPING IN XX
C  THE BEST KPTS VECTORS LOOKED AT SO FAR.
C
      DO 900 K=K1,NRANPT
  600    DO 650 I=1,NX
  650    X(I)=XMIN(I)+RAN2(SEED)*(XMAX(I)-XMIN(I))
         NRANX=NRANX+1
         IF(NRANX.GT.MXRANX) GO TO 1100
         CALL FEASBL(X,NX,X,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *              ,NMOVES,0)
         IF(NMOVES.GT.0) GO TO 600
         CALL FUNC(X,NX,F)
         IF(F.LE.FF(JMINF)) GO TO 900
         FF(JMINF)=F
         DO 700 I=1,NX
  700    XX(I,JMINF)=X(I)
         JMINF=1
         DO 800 J=2,KPTS
  800    IF(FF(J).LT.FF(JMINF)) JMINF=J
  900    CONTINUE
C
 1000 IOK=1
      RETURN
C
 1100 WRITE(6,3) NRANX
    3 FORMAT('0***** IN GBO, WE HAVE EXCEEDED THE MAX NUMBER'
     *      ,' OF RANDOM POINTS (',I6,' )')
      IOK=0
      RETURN
      END
```

```
      SUBROUTINE GB1(XX,NX,KPTS,X0,FF,XMIN,XMAX,XIMIN,XIMAX,XI
     *               ,NIC,IOK)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION XX(NX,KPTS),X0(NX),FF(KPTS),XMIN(NX),XMAX(NX)
      DIMENSION XIMIN(NIC),XIMAX(NIC),XI(NIC)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *               ,IPRINT,MAXMIN,SEED,NRANPT,MAXRAN
C
C  CREATES A RANDOM INITIAL COMPLEX, XX, AROUND AND INCLUDING
C  A GIVEN FEASIBLE VECTOR X0.
C
C  CHECK X0 FOR FEASIBILITY AND THEN PUT IT IN XX
C
      CALL FEASBL(X0,NX,X0,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *           ,NMOVES,0)
      IF(NMOVES.GT.0) GO TO 1000
      DO 50 I=1,NX
   50 XX(I,1)=X0(I)
      CALL FUNC(X0,NX,FF(1))
C
C  FILL REST OF XX WITH RANDOM VECTORS WHICH HAVE BEEN FORCED
C  FEASIBLE BY MOVING THEM TOWARDS X0 IF NECESSARY.
C
      NRANX=0
      MXRANX=MAXRAN*(KPTS-1)
      DO 400 J=2,KPTS
  100    DO 200 I=1,NX
  200    XX(I,J)=XMIN(I)+RAN2(SEED)*(XMAX(I)-XMIN(I))
         NRANX=NRANX+1
         IF(NRANX.GT.MXRANX) GO TO 1100
         CALL FEASBL(XX(1,J),NX,X0,XMIN,XMAX,XIMIN,XIMAX,XI,NIC
     *              ,NMOVES,N2CUTS)
         IF(NMOVES.GT.N2CUTS) GO TO 100
         CALL FUNC(XX(1,J),NX,FF(J))
  400    CONTINUE
C
      IOK=1
      RETURN
C
 1100 WRITE(6,2) NRANX
    2 FORMAT('0***** IN GB1, WE HAVE EXCEEDED THE MAX NUMBER'
     *      ,' OF RANDOM POINTS (',I6,' )')
      IOK=0
      RETURN
C
 1000 WRITE(6,1) X0
    1 FORMAT('0***** NON-FEASBIBLE INITIAL POINT GIVEN TO GB1',/
     *      ,' X0 =',1P10E12.4)
      IOK=0
      RETURN
      END
```

```
      SUBROUTINE FUNC(X,N,F)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(N)
      COMMON /BOXCOM/ STEP,RELTOL,ABSTOL,NTOL,N1CUTS,N2CUTS,NLOOPS
     *               ,IPRINT,MAXMIN
      DIMENSION Y(16)
      DATA Y/34780.,28610.,23650.,19630.,16370.,13720.,11540.,9744.,8261
     &     .,7030.,6005.,5147.,4427.,3820.,3307.,2872./
      RESQ(A,B,C,Y,T) = (Y - A*DEXP(B/(T+C)))**2
C
C  COMPUTES F, THE VALUE OF THE FUNCTION WHEN EVALUATED
C  AT THE POINT X (AN N-VECTOR).
C
      F = 0.
      DO 100 J=1,16
       T = 50.+5.*(J-1)
100      F = F + RESQ(X(1),X(2),X(3),Y(J),T)
      F = DSQRT(F)
C
      F=MAXMIN*F
      RETURN
      END




      SUBROUTINE IMPLCT(X,NX,XI,NIC)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION X(NX),XI(NIC)
C
C  COMPUTES XI, THE NIC-VECTOR OF IMPLICIT CONSTRAINT
C  FUNCTION VALUES AT THE NX-VECTOR X.
C
      XI(1) = X(2)/(50.+X(3))
C
      RETURN
      END
```

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ESD-TR-81-155	2. GOVT ACCESSION NO. AD-A104858	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Practical Aspects of Nonlinear Optimization		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER Technical Report 576
7. AUTHOR(s) Richard B. Holmes and Jeffrey W. Tolleson		8. CONTRACT OR GRANT NUMBER(s) F19628-80-C-0002
9. PERFORMING ORGANIZATION NAME AND ADDRESS Lincoln Laboratory, M.I.T. P.O. Box 73 Lexington, MA 02173		10. PROGRAM ELEMENT PROJECT, TASK AREA & WORK UNIT NUMBERS Program Element Nos. 63304A and 63398A
11. CONTROLLING OFFICE NAME AND ADDRESS Ballistic Missile Defense Program Office Department of the Army 5001 Eisenhower Avenue Alexandria, VA 22333		12. REPORT DATE 19 June 1981 13. NUMBER OF PAGES 58
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* Electronic Systems Division Hanscom AFB Bedford, MA 01731		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

nonlinear programming method

computer code

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A general purpose nonlinear programming method and computer code are presented. The method is basically heuristic but extremely simple and reliable. Interactive operation of the code and its performance on several test problems is described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*